Design of a AA Requester-Responder

Diego R. Lopez, Candido Rodriguez, José Manuel Macías

1. Purpose of the Design

One of the TF-AACE planned deliverables was the development of a *reference implementation* for AA systems. This reference implementation would help in quickly establishing operable infrastructures, and in validating component and/or infrastructure interoperability. The evolution of the technology and the availability of several solutions both in the academic and the commercial world have made obsolete the first objective of this reference development. On the other hand, this evolution makes more necessary the availability of interoperability assessment mechanisms. Whenever a new component is developed and has to be integrated in a given infrastructure, or two different infrastructures must somehow be connected, these assessment mechanisms must be in place.

The number of different components, systems, APIs, etc. able to run into AA interactions seems to be much greater in the near future, as other application domains become aware of the available technologies and their potential benefits. In many cases, existing procedures or specific constraints will drive to the establishment of "local" solutions that should be able to interoperate with national, international, or transnational initiatives. These components are going to operate at different levels upon the network infrastructure (and even inside), so many of the assumptions that have driven the development of current systems in what relates to software architecture, languages, etc. will not hold in a significative number of cases. Lastly, as it has been widely recognized, different infrastructures (supported by federations, virtual organisations, or whatever other frameworks) are going to have different requirements on which attributes and values are to be requested and/or enforced.

The purpose of the system described here (a Authentication and Authorisation Requester-Responder, AA-RR) is the use of some sort of metadata describing the requirements of a certain infrastructure in order to validate (or make at least an assessment on) the interoperability of a certain component with other(s). The availability of this system will translate into:

- Easier interoperability efforts: it is easier not to test against infrastructures in use, or avoid the requirement for installing additional software just for testing.
- A coherent collection of profile data, applicable not only to different pieces of software but to whole infrastructures.
- Simpler (con-)federation mechanisms, since requirements on syntax and semantics can be published and shared in a normalized way.

2. AA-RR functionality

The system will be built according to the following assumptions:

1. AA interactions are based on the (trusted) exchange of properties (that we'll call attributes) about users, either humans or applications, and resources.
2. There is a common consensus in using open standards to guide AA interactions, not only in the middleware NREN community, but also in the Grid community and the industry.

3. Any standard leaves much open issues, so different profles can be applicable to accomplish a certain task.

The AA-RR will be able to use specific definitions to simulate the external behavior of different components of several infrastructures in order to assess the interoperability of a certain element with them. This implies that the AA-RR can be used to learn of those elements it connects to, enhancing its knowledge base. There are three types of components that the AA-RR will be able to impersonate, each one corresponding to one of the components in a general AA architecture:

1. Attribute sources (like a Shibboleth AA, a A-Select server, a PAPI AAAS, or an Athens XAP). These are, esentially, entities able to accept attribute queries from attribute requesters (entities of the second type), validate the queries according to their privacy-protection rules, and respond with attribute information.

2. Attribute requesters (like a Shibboleth SHAR, a VOMS server, a PAPI PoA, or a Athens DSP entry point). These entities perform requests about user attributes to attribute sources (entities of the first type) and make an authorisation decision on them, possibly querying an authorisation engine (an entity of the third type).

3. Authorization engines (like Permis or SPCOP). These entities make decisions from the requests they receive from attribute requesters (entities of the second type) and their internal configuration. They return a simple (yes/no) or complex (for example, a SPOCP "blob") answer to the query.

The broad application fields to which AA interactions are aimed implies a diverse variety of potential protocols to be used, specially if we take into account that currently none of the proposed standards is clearly dominant over the others. Therefore, the AA-RR will be built using a protocol-agnostic approach, providing a neutral format for defining its behaviour, and using open libraries to implement protocol bindings. At least three of these bindings will be considered:

- XML-based protocols: SAML (and XACML) over SOAP/HTTP.

- The SPOCP protocols, either for authorization engine queries or attribute requester/source interactions.

- RADIUS and/or its derivatives, for attribute requester/source interactions.

3. The AA-RR Architecture

As said in the previous section, it is assumed that AA interactions consist of queries for data and/or authorisation decisions and responses to that queries. Each query contains either a set of attributes to retrieve, or a set of attribute/value pairs to decide about. Responses contains attribute/value pairs in response to an attribute query, or data relative to authorisation. Queries and responses may make use of mechanisms to establish trust between components, such as X.509 certificates and shared keys. The actual aspect of these interactions will depend, obviusly, on the protocol binding to be applied. The AA-RR behaviour will be controlled by a set of rules that specify which queries/responses will be accepted and/or sent, which attributes and values will be requested/asserted, which error conditions will be detected, and which final results will yield according to the evolution of the interaction.

The AA-RR will be able to load a profile defining which role (of the three described above) has to play for a certain session, and the relevant parameters governing its behaviour:

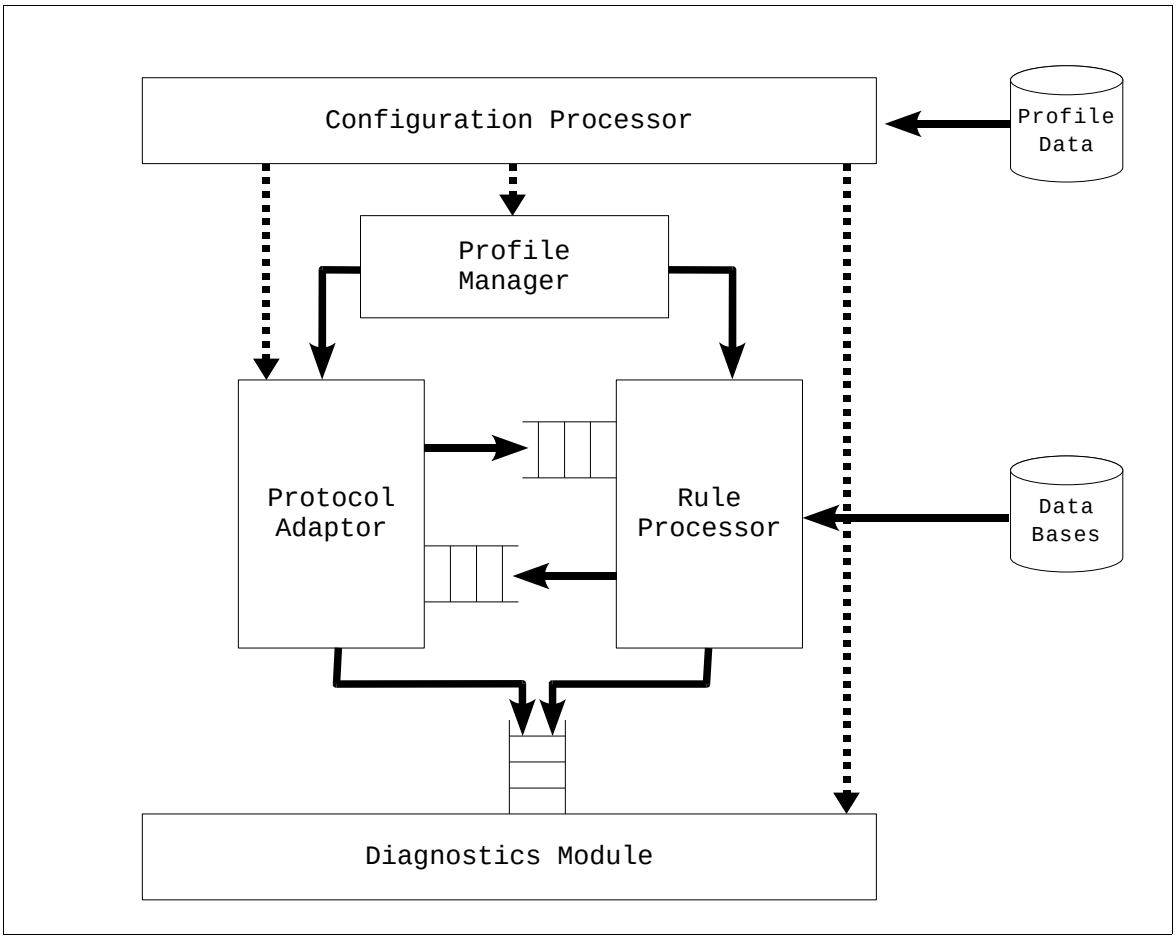
- Protocol binding to be applied.

- Configuration of the protocol binding: ports, trust elements (certificates, keys,...), protocol options,...

- Environment settings (files, log level, data bases).

- Rules controlling the actual evolution of the AA interactions.

The following figure depicts the proposed architecture of the AA-RR system:



The functions of each of the components are as follow:

- The *Configuration Processor* reads profile data and instantiates the required components, including the applicable protocol binding.
- The *Profile Manager* controls the execution of the different elements in the profile, directly starting the requesters or intiating the responders for the required AA interactions.
- The *Rule Processor* applies the rules defined to specify the behaviour of the AA-RR for the required interactions.
- The *Diagnostic Module* logs information about the running interactions and about their results.
- The *Protocol Adaptor* provides the other components a uniform interface to the different protocol bindings.

3.1. Configuring the AA-RR and Executing a Profile

The configuration of the AA-RR will be made by means of XML files, that define the profile to be applied and the defined behaviours by means of *rulesets*. The general structure of a AA-RR configuration file will consists of a single element, called `aarrdef`, that contains the mandatory attribute `binding`, defining the protocol binding to be applied. According to the protocols bindings described above, the following elements will be accepted:

- `<aarrdef binding="saml">` for SAML over SOAP/HTTP.
- `<aarrdef binding="spocp">` for SPOCP.
- `<aarrdef binding="radius">` for RADIUS.

The contents of this element consists of three sections, each one defined by a corresponding element:

1. The *Protocol configuration* section, defined by the optional element `protocol`, that will contain protocol-specific definitions. Each of these definitions will be held by the corresponding element and its attribute. Since this section is protocol specific, its contents will depend on the binding being applied, and a complete definition of them (and the applicable default values) will be available as bindings are incorporated into the system. The prototype system on which this design is based (built for the `saml` binding) currently accepts the following elements:

    - `<listen at='URL'/>`, the URL the AA-RR will be contacted through (when accepting connections).

    - `<connect to='URL'/>`, the URL the AA-RR will connect to (when requesting connections).

    - `<tlspeerid required='always|never|onrequest'/>`, specifying how TLS-based peer identification will be validated, when applicable.

    - `<xmlsign required='always|never|onrequest'/>`, specifying how XML signature verification and procedures will be applied.

    - `<keystore path='PathToFile'/>`, a keystore holding the necessary private keys and certificates to process cryptographic requests when applicable.

2. The *Environment configuration* section, defined by the element `environment`, that will contain the following elements:

    - `<instance type='TypeOfEntity'/>`, defining the type of entity to be emulated by the AA-RR: a responder (`responder`) or a requester (`requester`).

    - `<workingdir path='PathToDirectory'/>`, defining the absolute path where other files will be read and/or written.

    - `<log level='NumericValue' output='OutputMethod'/>`, specifying the detail to which events will be logged during the execution of the AA-RR, and the method to perform it (file, standard I/O descriptor,...).

    - `<base name='ReferenceName' path='PathToFile'/>`, defining a name (to be used in references) and a path (relative to the working directory) where data bases containing values to be applied during the AA-RR executions can be found. The only currently supported format for this file will be a collection of lines, in which the first word (set of non-blanks characters) will act as a key, and the rest of the line as the value for that key.

    - `<timeout seconds='IntegerNumber' miliseconds='IntegerNumber'>`, defining the value of the wait timeouts to be applied when rulesets are executed.

3. The *Rulesets* definition section, defined by the element `run`, that will contain a non-empty list of `ruleset` elements. The contents of the `ruleset` element will be discussed in detail in the section describing the application of rules. For the moment, it is worth mentioning that, inside the main AA-RR configuration file, this element may take two forms:

    (a) `<ruleset name='RulesetName' path='PathToFile'/>`, indicating that the ruleset is stored inside a different file (defined by the relative path provided in the `path` attribute). This file has to contain a `ruleset` element by its own.

    (b) `<ruleset name='RulesetName'> ...STATES... </ruleset>`. In this case, the ruleset is defined inside the current element.

Once the configuration processor has loaded and parsed the configuration file, and all the components are appropriately instantiated, the profile manager will take control of the AA-RR, initializing and connecting the event queues of the rule

processor and the protocol adaptor, and giving both modules access to the diagnostic queue. Rulesets are executed sequentially, according to the order in which they are referenced in the configuration file. The current execution of the AA-RR finishes when the last ruleset has been run.

3.2. Applying Rules

As stated above, rulesets are defined by means of the `ruleset` element. This element uses the mandatory attribute `name` to identify the ruleset, so it can be better referenced and accessed (in diagnostics logs, source exchanges, library maintenances, etc.). In the general configuration file, a second attribute (`path`) can be used to point the configuration processor to load the actual contents of the ruleset from a separate file: in this case the ruleset definition, as it directly appears in the main configuration file, will be empty.

A ruleset definition consists of a non-empty list of `state` elements, which have the attribute `name`, used to assign an identifier to the state, so it can be directly accessed from other states in the ruleset. Only one state is *active* at a given moment. The initial state for a ruleset is the first that is found in the ruleset definition. The contents of a `state` element consist of a non-empty list of `rule` elements.

A `rule` element can have an optional `name` attribute and contain a (single and optional) `conditions` element and a (single and mandatory) `actions` element.

An `actions` element defines a reaction of the AA-RR and consists of a list of `action` elements. Each `action` element may contain the attribute `name` (an identifier for further referencing the action), and one of the following attributes:

- `finish`, indicating that the ruleset must finish issuing the result defined as the value of the attribute. The acceptable values are `pass`, `fail` and `inconclusive`. After finishing, the system will return to an initial state.

- `exit`, indicating that the ruleset must finish (as in `finish`) but the system must also stop completely in stead of returning to its initial state. The aceptable values are also `pass`, `fail` and `inconclusive`.

- `next`, identifying a state (through its `name` attribute) that will be the next active one.

- `send`, identifying a protocol data unit (through a name defined by the protocol adaptor in use) to be sent. The contents of the action element define further characteristics for the data to be sent: an empty action element with a `send` attribute will make the protocol adaptor to send a default data unit. Optional attribute `src` can be used to specify an external skeleton file to be used for building a response. A non-empty action element contains a list of field elements, with the following attributes:

    - `id`, which identifies the name of the component using a value defined by the protocol adaptor.

    - `value`, which allows for the provision of a direct value. The prefix 0x can be used for providing binary values in hexadecimal format.

    - `base` and `key`, used to identify a value defined by the corresponding key and data base.

The `conditions` element is used for controlling how the AA-RR behaves according to the events it detects, and consists of a list of `condition` elements. All the constraints defined inside a conditions element must be satisfied in order to fire the rule that contains it (that is, there is an implicit and composition of conditions inside a given element). A rule with no conditions is always applicable.

A `condition` element may contain the attribute `name` (an identifier for further referencing the condition), and one of the following elements:

- `receive`, identifying a protocol data unit (through a name defined by the protocol adaptor in use) that satisfies the condition if it is received by the AA-RR.

- `field`, identifying a component of the received protocol data unit (using a value defined by the

protocol adaptor). If this attribute is present the condition is satisfied whenever:

- The component is present, and
- Its value corresponds to the content of the `value` attribute or the contents defined by the attributes `base` and `key` (as defined for actions above), if any of this attributes are defined.

- `default`, that allows the condition to be satisfied in a general case. If present, this attribute may take the values `any` (any event matches), `timeout` (for timeout events) or `exception` (exceptions signaled by the protocol adaptor).

Inside a state, rules are tested in sequential order: if a rule fails, the next inside the state is tried. The rule processor waits on a state until one of the alternative rules is fired.

As an example, consider the following ruleset definition:

```
<ruleset name="SAMLAQ-eduPersonEntitlement-role">
 <state name="init">
  <rule name="i1">
   <conditions>
    <condition name="c1" receive="SAMLAttributeQuery"/>
    <condition name="c2" field="Attribute" value="eduPersonEntitlement"/>
    <condition name="c3" field="Attribute" base="attNames" key="roleAttr"/>
   </conditions>
   <actions>
    <action name="a1" send="SAMLAttributeResponse">
     <field id="AttributeValue"
      value="urn:mace:rediris.es:entitlementForPAPIResources"/>
     <field id="AttributeValue" base="attValues" key="roleAttr"/>
    </action>
    <action name="a2" next="accepted"/>
   </actions>
  </rule>
  <rule name="idef">
   <conditions>
    <condition name="ca1" default="any"/>
   </conditions>
   <actions>
    <action name="ff" finish="fail"/>
   </actions>
  </rule>
 </state>
 <state name="accepted">
  <rule name="r2">
   <actions>
    <action name="fp" finish="pass"/>
   </actions>
  </rule>
 </state>
</ruleset>
```

Will yield a `pass` result when a `SAMLAttributeQuery` requesting the attributes `eduPersonEntitlement` and that defined by the key `roleAttr` in the data base `attNames`, after sending a `SAMLAttributeResponse` containing the corresponding values. If any other event is received, the ruleset will yield a `fail` result.